NADIA Workbook

ARC Centre of Excellence for Coherent X-Ray Science

T'Mir Julius

T'Mir Julius

July 5, 2013

Contents

1 Introduction

1.1 Coherent Diffractive Imaging

Coherent Diffractive Imaging (CDI) is a technique used to image nanoscale objects. CDI is a lensless approach and so avoids the aberrations that result from the transmission and refraction of electromagnetic waves through a lens. Instead, a highly coherent incident wave is allowed to scatter off an object to be imaged, and the resulting diffraction pattern is recorded by a detector. An image of the object can then be recovered by propagating the wave at the detector plane back to the object plane using an inverse Fourier transform. However, it is not possible to record all the information of the diffraction pattern as the detector is able to record only the amplitude of the scattered wave, and not its phase. As a result, we are left with only part of the information required to recover an image of the object. In order to get a full image we need to recover the phase information.
To do this, we use an iterative procedure. A random phase is assigned to the diffraction pattern, and an inverse Fourier transform is applied to propagate the wave to the object plane. Information about the maximum size of the object is applied, and the result is propagated back to the detector plane using a Fourier transform. This new wave is rescaled to match the original detector reading. This gives a diffraction pattern with a new phase, but the same amplitude as that observed in the detector. This process is repeated until the iterations converge and from this an image of the original object is found. The only feasible way to perform this iterative procedure is computationally.

1.2 NADIA

NADIA aims to be a user-friendly, robust and flexible package that contains well-tested libraries that make it easier for researchers to access and implement the various CDI algorithms on data. This is directed at new researchers entering the field as well as more experienced researchers looking to get results from data, experiment with a new technique without investing significant time and resources in writing up the code required, or even just to perform a cross-check with results already achieved. A number of different algorithms have been implemented and are described in section 2. We are trying to make NADIA a resource where these methods can all be found in one place, making it easier to get a new project off the ground.

2 Iterative Algorithms

2.1 Iterative Solution of Paraxial Wave Equations

The approach that is most often adopted in coherent diffraction imaging is to seek an iterative solution of the paraxial wave equation by propagating modal envelope functions, $\psi_n(\mathbf{r}_\perp, z, \omega)$ between two parallel planes at $z = z_1$ and $z = z_2$. In the case of quasi-monochromatic light of full spatial coherence we may drop both the modal index, n, and the frequency label, ω, defining the coherent planar wavefield $\psi_c(\mathbf{r}_\perp, z)$. The plane at z_1 is usually denoted the object plane, though it really represents the exit surface of the object that imparts constraints on the free-space propagation of the wavefield to the detector plane at z_2. In this plane, we have measurements of the intensity, which are typically obtained as discrete samples on a rectilinear grid using, for example, a charge-coupled device camera. The measurement of the intensity is used as a constraint on $|\psi_c(\mathbf{r}_{\perp,2}), z_2|$ in the case of coherent wavefields, or on the sum of the separate modal intensities and, consequently, their individual amplitudes in the case of a partially-coherent quasi-monochromatic wavefield. The general strategy in iterative solutions of paraxial wave equations is to propagate planar envelope functions between these two planes, satisfying whatever *a priori* information is available until the iteration becomes self-consistent. In this regard, the procedure is analogous to the self-consistent solution of the Hartree-Fock

equations. In the Hartree-Fock equations, which have long served as the cornerstone of theoretical atomic, molecular and solid-state physics and chemistry, a trial form of a quantum mechanical wavefunction is used to define an effective electrostatic mean-field potential. This is, in turn, used to generate an orthonormal orbital basis through the solution of coupled one-particle Schrödinger equations. This orbital basis is then used to construct an updated many-electron wavefunction for the entire system, which in turn specifies a new electrostatic Hartree-Fock potential that continues the procedure into a new basis, and a new iteration. The procedure is repeated until the output of a given iterate, the many-electron wavefunction or, equivalently, the orbital basis from which it constructed, is deemed to be negligibly different from the corresponding quantities that formed the input of the iteration. Throughout the procedure, the solution is constrained by the presence of a fixed external Coulomb potential in the Hamiltonian, a given number of electrons, and a specified electronic configuration.
Iterative solutions of the paraxial wave equations in imaging applications share many of the features, both good and bad, of the Hartree-Fock equations in electronic structure theory. They are rather simple to implement; using readily-available code fragments one may readily assemble a simple computer program to perform the necessary procedures with very little effort. Common to both problems, however, is the issue of erratic convergence properties of these iterative schemes. From a random initial configuration, the non-linear, non-convex nature of the optimisation problems being solved places these schemes outside the domain of elementary mathematical optimisation problems. The iterative solution of wave equations in imaging applications possess, as we shall see, the additional difficulty that the updating of one approximation to the next is discontinuous; numerical approximations of the Hartree-Fock equations are amenable to linearisation and to the construction of starting guesses that are close to the desired solution. Much of the development of iterative methods in imaging applications in recent years has been directed to investigations of these convergence difficulties.

2.1.1 Vector spaces and projection operators

We assume that the solution we seek may be written as a vector, $\mathbf{x}$, whose elements are real numbers, $\{x_k\}$. From a physical point of view, these numbers represent discrete samples of a complex wavefield or, more generally, the complex amplitudes of modal functions used to expand the mutual optical intensity associated with a partially-coherent wavefield. Assuming that each wavefield or mode is sampled on a rectilinear grid of dimension $N \times N$ and that the number of active modes is M, the length of the parameter vector is $2MN^2$; the factor of two reflects the presence of the real and imaginary parts of the desired physical quantities. In the special case of a quasi-monochromatic wavefield possessing full spatial coherence, we recover conventional formulations of coherent diffractive imaging by setting $M = 1$.
The norm of the parameter vector, $\mathbf{x}$, is denoted $||\mathbf{x}||$ and is defined by

$$||\mathbf{x}|| = \left\{\mathbf{x}^\dagger \mathbf{x}\right\}^{1/2} = \left\{\sum_k x_k^2\right\}^{1/2} . \tag{1}$$

The set of possible solution vectors defines a metric space, in the sense that the distance between any two such vectors may be assigned the $2MN^2$-dimensional interpretation

$$||\mathbf{x}_1 - \mathbf{x}_2|| = \left\{\sum_k (x_{2,k} - x_{1,k})^2\right\}^{1/2} . \tag{2}$$

Most imaging algorithms implement constraints on $\mathbf{x}$ through the action of operator representations of two types of information; information about the spatial extent of the scattering target in the object plane, and knowledge of the intensity of the wavefield in the detector plane. The support operator, $\hat{P}_S$, has the effect that

$$\begin{aligned} \hat{P}_S\, \mathbf{x} \quad &\mapsto \quad x_k \qquad \text{for } k \in S, &(3)\\ &\mapsto \quad 0 \qquad \text{for } k \ni S &(4) \end{aligned}$$

where S denotes the spatial domain of the object, which is associated with the vector labels k in a discrete representation. This has the effect that the real and imaginary parts of any wavefield or mode are set to zero at sample points outside the object boundaries and are left untouched inside the object boundaries under the action of $\hat{P}_S$ on $\mathbf{x}$.
The amplitude projector, $\hat{P}_F$, rescales the amplitudes of a coherent wavefield, or of the coherent modes in a partially-coherent system, based on experimental intensity measurements. This is often called the Fourier modulus constraint in coherent diffractive imaging, through we will use it in a more general context here. The amplitude constraint is defined by

$$\hat{P}_F\ \mathbf{x} \quad \mapsto \hat{D}_{-Z}\hat{R}\hat{D}_Z\ \mathbf{x} \tag{5}$$

where $\hat{D}_Z$ represents free-space propagation of each mode by a distance Z and $\hat{D}_{-Z} = \hat{D}_Z^{-1}$. The rescaling operator $\hat{R}$ enforces the condition that the intensity associated with $\hat{D}_Z\mathbf{x}$ corresponds, pointwise, to $I(\mathbf{r}_\perp)$, the measured intensities in the detector plane.
In the single-mode formulation that is conventionally used in coherent diffractive imaging,

$$\left[\hat{R}\hat{D}_Z\ \mathbf{x}\right]_K = \sqrt{\frac{I_K^0}{I_K}}X_K \tag{6}$$

where $X_K = \left[\hat{D}_Z\ \mathbf{x}\right]_K$, $I_K^{1/2} = |X_K|$ and K labels the N^2 measured sample intensity values, I_K^0. The Fresnel free-space propagator reduces to a simple instance of the Fourier transform operator in the limiting base of far-field diffraction. The effect of $\hat{R}$ on $\hat{D}_Z\mathbf{x}$ is to preserve the phase of X_K while rescaling its amplitude to correspond to a wavefield whose measured intensity is I_K^0 in the detector plane. The extension to several coherent modes is straightforward; I_K is evaluated as the incoherent sum over modal contributions, $I_{K,m}$, and $X_{K,m}$ refers to a sample of a forward-propagated mode labelled m, rather than a coherent wavefield. This preserves the phase of each mode under free-space propagation, while rescaling the amplitude of every mode at sample point K by a single scaling factor, $\sqrt{I_K^0/I_K}$.
The vectors that satisfy $\hat{P}_S\ \mathbf{x} \mapsto \mathbf{x}$ define the set $\mathcal{S}$, while the vectors that satisfy $\hat{P}_F\ \mathbf{x} \mapsto \mathbf{x}$ define the set $\mathcal{M}$. Solution algorithms seek a vector $\mathbf{x}$ that is at the intersection of the sets, $\mathbf{x} \in \mathcal{S} \cap \mathcal{M}$, on the assumption that this intersection contains a single vector, or a finite number of trivially-related vectors that may be derived from one another by symmetry transformations.
The sets $\mathcal{S}$ and $\mathcal{M}$ have distinct mathematical properties that complicate the determination of the required point of intersection. The vectors belonging to the set $\mathcal{M}$, may be visualised, for each K, as comprising circles in the complex plane whose radii are $\sqrt{I_K^0}$. A linear combination of two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, that are each in $\mathcal{M}$ is not, in general, also in $\mathcal{M}$ and, in particular

$$|t\mathbf{x}_1 + (1-t)\mathbf{x}_2|_K \ni \mathcal{M} \tag{7}$$

for all $0 \leq t \leq 1$. In practice, this means that we cannot continuously transform between $\mathbf{x}_1$ and $\mathbf{x}_2$, which are both in $\mathcal{M}$, and be certain that we remain in $\mathcal{M}$ throughout. The set $\mathcal{M}$ is consequently described as being non-convex, and the operator $\hat{P}_F$ is non-linear, since

$$\hat{P}_F(\mathbf{x}_1 + \mathbf{x}_2) \neq \hat{P}_F\mathbf{x}_1 + \hat{P}_F\mathbf{x}_2.$$

It is the non-convexity of $\hat{P}_F$ that causes iterative schemes that employ it to be prone to stagnation. On the other hand, any linear combination of vectors in $\mathcal{S}$ is also a member of $\mathcal{S}$, which is described as convex. The operator $\hat{P}_S$ is linear, since

$$\hat{P}_S(\mathbf{x}_1 + \mathbf{x}_2) = \hat{P}_S\mathbf{x}_1 + \hat{P}_S\mathbf{x}_2.$$

Despite these difficulties, the formulation and discussion of iterative phase retrieval problems has, nevertheless, largely been conducted in the language of projections onto convex sets, for which there exists a considerable literature [4, 5].

2.1.2 Contraction mappings

The determination of the required solution vector may be regarded as being equivalent to the solution of non-linear equations for its elements. A mapping, $g(\mathbf{x}_n)$, may be devised that operates on the set of vectors, $\{\mathbf{x}_n\}$, whose components are real numbers, and whose dimension corresponds to the dimension of the solution vector, $\mathbf{x}$. Let us assume that we are able to identify some relation, $C(\mathbf{x}_k)$, that possesses the property $C(\mathbf{x}) = 0$, that is satisfied uniquely by the target solution vector, $\mathbf{x}$. In order to define an iterative procedure to determine $\mathbf{x}$, we employ the mappings

$$\begin{aligned} \mathbf{x}_{k+1} &= g(\mathbf{x}_k) && (8) \\ g(\mathbf{x}_k) &= \mathbf{x}_k - \mathbf{M}(\mathbf{x}_k)C(\mathbf{x}_k). && (9) \end{aligned}$$

We identify the target solution as the fixed point of the mapping for which $C(\mathbf{x}) = 0$ and $\mathbf{x} = g(\mathbf{x})$. The matrix $M(\mathbf{x}_k)$ may be selected, or constructed, in such a way as to control the convergence characteristics of the scheme; in the absence of any reason to do otherwise, one typically chooses $M(\mathbf{x}_k) = \beta\mathbf{I}$, where $\mathbf{I}$ is a unit matrix and β is regarded as an iterative relaxation parameter. The iterative procedure defined by this scheme will converge to the desired solution once the trial vectors enter a neighbourhood of $\mathbf{x}$, $\mathcal{N}_\varrho(\mathbf{x})$, for which $\mathcal{N}_\varrho(\mathbf{x}) = \{\mathbf{x_k} : ||\mathbf{x}_k - \mathbf{x}|| \le \varrho\}$. The fixed positive constant, ϱ, defines an effective radius of the vector space over which the iteration is deemed to be converging to $\mathbf{x}$.
If $g(\mathbf{x}_k)$ possesses the property that,

$$||g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)|| \le \mu||\mathbf{x}_{k+1} - \mathbf{x}_k|| \tag{10}$$

for all $\mathbf{x}_k \in \mathcal{N}_\varrho(\mathbf{x})$ and $0 \le \mu \le 1$ and $\mathbf{x}_k$, then it is said to be a Banach contraction mapping, and the sequence converges to $\mathbf{x}$. The use of contraction mappings has been employed widely in the solution of systems of non-linear equations [15, 14].
If $\mathbf{x}_k$ is not in $\mathcal{N}_\varrho(\mathbf{x})$ then the trajectory of iterates need not be towards the solution in any geometrical sense. Elser [8, 9] has noted that iterative procedures of this general form are observed to follow an apparently chaotic trajectory, and the set that is searched behaves as if it were a strange attractor. The available empirical evidence in imaging applications suggests that there is typically only a single strange attractor, the consequence of which is that the space of vectors being searched is independent of the randomly-assigned initial conditions. The determination of solutions then behaves like a Poisson-distributed random process, eventually generating a sequence of vectors that is in $\mathcal{N}_\varrho(\mathbf{x})$. At that point, convergence to $\mathbf{x}$ can be quite rapid, with the rate of convergence governed by the parameter μ appearing in the contraction mapping, Eq. 10. The required solution is a fixed point attractor; the concept of a neighbourhood serves to define a set of starting vectors from which one can reasonably expect to see rapid convergence to $\mathbf{x}$. It is the chaotic nature of the trajectory through the vectors $\{\mathbf{x}_k\}$ from a random initial guess that is outside $\mathcal{N}_\varrho(\mathbf{x})$ that proves to be such a powerful tool in providing an efficient search algorithm for $\mathbf{x}$ from a random starting vector. The trajectory through the vector space, however, depends on the construction of $g(\mathbf{x}_k)$ and, in particular, on the selection of the form of both $C(\mathbf{x}_k)$ and $\mathbf{M}(\mathbf{x}_k)$. We now consider some of the most widely-used choices for these mappings that have been employed in iterative imaging algorithms.

2.1.3 Error reduction algorithm

The error-reduction (ER) algorithm is the simplest example of an iterative scheme that utilises projections onto non-convex sets to determine a solution at the intersection of the sets $\hat{\mathcal{F}}$ and $\mathcal{S}$. In the form formulated by Fienup [10, 11], the error reduction algorithm, also known as the Gerchberg-Saxton-Fienup algorithm [13], may be defined by the iterative sequence

$$\mathbf{x}_{k+1} = \hat{P}_S\hat{P}_F\ \mathbf{x}_k, \tag{11}$$

which involves alternating set projections. Algorithms of this type have been widely used in cases in which the two constraint sets, $\mathcal{A}$ and $\mathcal{B}$, are convex, in which case the sequence of the form defined by Eq. 11 will

converge to $\mathcal{A} \cap \mathcal{B}$. The non-convex nature of $\mathcal{M}$ invalidates any such assumption about the convergence of this sequence in the imaging applications considered here. A discussion of the use of iterative phase retrieval methods in imaging applications from the perspective of convex optimisation theory has been presented by Bauschke *et al.* [4, 5].
Starting from an initial vector, $\mathbf{x}_0$, a numerical implementation of this algorithm requires two instances of a Discrete Fourier transform procedure to generate $\mathbf{x}_1$. The algorithm defined by Eq. 11 is then repeated until $\mathbf{x}_{k+1} = \mathbf{x}_k$. While this corresponds to a fixed point of the mapping defined by Eq. 11, it need not correspond to a solution in $\mathcal{M} \cap \mathcal{S}$. While it is certain, by construction, that $\mathbf{x}_k \in \mathcal{S}$ for all $k \geq 1$, it is not necessarily the case that $\hat{P}_F \mathbf{x}_k = \mathbf{x}_k$; in fact, $\hat{P}_F \mathbf{x}_k$ may generate a vector that violates the support constraint, and in practice there are very many such vectors with this property that also satisfy $\mathbf{x}_{k+1} = \mathbf{x}_k$. In this case, the iteration is said to have stagnated; stagnation is a likely outcome when the ER algorithm is initiated by a random starting vector.
The ER algorithm remains, however, very useful, despite the tendency towards iterative stagnation. It is frequently used in conjunction with other algorithms that have generated a suitable starting vector that is an element of the neighbourhood governing its effective radius of convergence. Eq. 10, we may write the ER algorithm in the equivalent form

$$g(\mathbf{x}_k) = \mathbf{x}_k - \beta \mathbf{I} \left[\hat{P}_S \hat{P}_F - \hat{I} \right] \mathbf{x}_k \tag{12}$$

in which we may make the identification $C(\mathbf{x}_k) = \left[\hat{P}_S \hat{P}_F - \hat{I} \right] \mathbf{x}_k$ and $\mathbf{M}(\mathbf{x}_k) = \beta \mathbf{I}$, including a relaxation parameter, β, to control the rate of convergence. In fact, there is always some neighbourhood within which this simple and computationally efficient scheme will correspond to a contraction mapping, converging to $\mathbf{x}$ from a vector $\mathbf{x}_0$ within $\mathcal{N}_\varrho(\mathbf{x})$. It is also the case that the rate of convergence to $\mathbf{x}$ using Eq. 11 or Eq. 12 can be superior to any more elaborate scheme once a suitable vector to initialise the ER algorithm has been identified.

2.1.4 Basic Input-Output and Basic Output-Output algorithms

Fienup appears to have been the first to note that the iterative solution of imaging problems using intensity and support information shared many of the essential features encountered in signal processing applications. He performed a detailed comparison of possible solutions to the stagnation problems associated with the error-reduction algorithm which has become one of the most influential articles in this field [11]. In particular, Eq. 42 in [11]) in our notation, it is written as

$$g(\mathbf{x}_k) = \mathbf{x}_k - \beta \left[(\hat{I} - \hat{P}_s) \hat{P}_F \right] \mathbf{x}_k. \tag{13}$$

This is referred to as the Basic Input-Output (BIO) algorithm. If the input vector, $\mathbf{x}_k$, is chosen to be $\hat{P}_F \ \mathbf{x}_{k-1}$ for some iteration, k, then the iteration possesses the special property that $g(\mathbf{x_k}) = \mathbf{x}_k$, irrespective of the choice of $\mathbf{x}_{k-1}$. These fixed points of the iteration will not, in general, correspond to the target solution.
Similarly, one may replace the vector $\mathbf{x}_k$ by $\hat{P}_F \mathbf{x}_k$ on the right-hand side of the basic input-output algorithm, which Fienup writes in the form

$$g(\mathbf{x}_k) = \hat{P}_F \mathbf{x}_k - \beta \left[(\hat{I} - \hat{P}_s) \hat{P}_F \right] \mathbf{x}_k. \tag{14}$$

This is referred to by Fienup as the Basic Output-Output (BOO) algorithm.

2.1.5 Hybrid Input-Output algorithm

The hybrid input-output (HIO) algorithm is

$$g(\mathbf{x}_k) = \mathbf{x}_k + \left[(1+\beta) \hat{P}_S \hat{P}_F - \beta \hat{P}_F - \hat{P}_S \right] \mathbf{x}_k \tag{15}$$

in which we can make the identification $C(\mathbf{x}_k) = \left[(1+\beta)\hat{P}_S\hat{P}_F - \beta\hat{P}_F - \hat{P}_S\right]\mathbf{x}_k$ and $\mathbf{M}(\mathbf{x}_k) = -\mathbf{I}$. In this case, $C(\mathbf{x}_k)$ contains a real and, apparently, arbitrary parameter, β, but it still satisfies the fundamental condition for a satisfactory solution, $C(\mathbf{x}) = 0$. We could, of course, multiply $\mathbf{M}(\mathbf{x}_k)$ by a second relaxation parameter in order to change the rate of convergence of the iteration, or to influence the trajectory through the search space far from convergence.

2.1.6 Difference map algorithm

The difference-map (DM) algorithm of Elser [8] is

$$g(\mathbf{x}_k) = \mathbf{x}_k + \beta\left[\hat{P}_s\left(\hat{P}_F - \frac{1}{\beta}(\hat{P}_F - \hat{I})\right) - \hat{P}_F\left(\hat{P}_S + \frac{1}{\beta}(\hat{P}_S - \hat{I})\right)\right]\mathbf{x}_k. \tag{16}$$

This algorithm contains a combination of terms involving $\hat{P}_s\hat{P}_F\,\mathbf{x}_k$, $\hat{P}_F\hat{P}_S\,\mathbf{x}_k$, $\hat{P}_S\,\mathbf{x}_k$, $\hat{P}_S\,\mathbf{x}_k$, $\hat{P}_F\,\mathbf{x}_k$ and $\hat{I}\,\mathbf{x}_k$, with the combinations exhibiting a high degree of symmetry. Taken together, one may identify $\mathbf{M}(\mathbf{x})C(\mathbf{x}_k)$ with the second term on the right-hand side of Eq. 16 and $\mathbf{M}(\mathbf{x}_k) = -\beta\mathbf{I}$. Unlike the ER and HIO algorithm, however, $C(\mathbf{x}_k)$ is constructed from several subunits that vanish independently in the limit $\mathbf{x}_k \to \mathbf{x}$, which becomes more apparent if we write $C(\mathbf{x}_k)$ in the form

$$C(\mathbf{x}_k) = \left[\left(\hat{P}_F\hat{P}_S - \hat{P}_S\hat{P}_F\right) - \frac{1}{\beta}\left(\hat{P}_S\left(\hat{P}_F - \hat{I}\right)\right) - \frac{1}{\beta}\left(\hat{P}_F\left(\hat{P}_S - \hat{I}\right)\right)\right]\mathbf{x}_k \tag{17}$$

The empirical evidence presented by Elser *et al.* [9] regarding the operation of the difference-map algorithm suggests that the construction of an operator consisting of all operator products of $\hat{P}_S$ and $\hat{P}_F$ causes chaotic evolution of $\mathbf{x}_k$ for vectors that are far from $\mathbf{x}$. The presence of the combination of the three operators that vanish in Eq. 17 when $\mathbf{x}_k \to \mathbf{x}$ also strongly suggests, as Elser discusses, that the iteration is rather insensitive to the presence of the solution except when it enters the neighbourhood within which the mapping performs sequential contractions of $\mathbf{x}_k$ towards $\mathbf{x}$.

2.1.7 A more unified description of iterative algorithms

The review by Marchesini [17] provides comparisons of available iterative projection methods with conventional non-linear optimisation techniques, including the method of steepest descents, the conjugate gradient method, and a saddle-point optimisation scheme [16] that may be regarded as a extension of the hybrid input-output algorithm. Rather than regard the available phase retrieval schemes as representing an unrelated algorithmic zoology, to be catalogued and categorised, we seek here a more unified perspective of the relations between them.
The matrix-vector product, $\mathbf{M}(\mathbf{x}_k)C(\mathbf{x}_k)$ that appears in Eq. 10 must generate a vector that is commensurate with $\mathbf{x}_k$. It is not necessary, however, that $C(\mathbf{x}_k)$ is commensurate with $\mathbf{x}_k$, which we exploit in constructing a general formulation of iterative methods.
The iterative projective algorithms that appear in the published literature can all be rewritten in terms of a basis involving only five intermediate quantities, which are $\hat{P}_S\hat{P}_F\,\mathbf{x}_k$, $\hat{P}_F\hat{P}_S\,\mathbf{x}_k$, $\hat{P}_S\,\mathbf{x}_k$, $\hat{P}_F\,\mathbf{x}_k$ and $\hat{I}\,\mathbf{x}_k$. From these, we may construct a basis of ten vectors, $\mathbf{v}_m(\mathbf{x}_k),\ m = [1, 10]$, each of which is commensurate with $\mathbf{x}_k$, and each of which vanishes when $\mathbf{x}_k = \mathbf{x}$. These vectors are

$$\begin{aligned}
\mathbf{v}_1(\mathbf{x}_k) &= \left[\hat{P}_S\hat{P}_F - \hat{P}_F\hat{P}_S\right]\mathbf{x}_k,\\
\mathbf{v}_2(\mathbf{x}_k) &= \left[\hat{P}_S\hat{P}_F - \hat{P}_F\right]\mathbf{x}_k,\\
\mathbf{v}_3(\mathbf{x}_k) &= \left[\hat{P}_S\hat{P}_F - \hat{P}_S\right]\mathbf{x}_k,\\
\mathbf{v}_4(\mathbf{x}_k) &= \left[\hat{P}_S\hat{P}_F - \hat{I}\right]\mathbf{x}_k,
\end{aligned}$$

Algorithm	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
ER	0	0	0	1	0	0	0	0	0	0
BIO	0	β	0	0	0	0	0	0	0	0
BOO	0	β	0	0	0	0	0	0	1	0
HIO	0	β	1	0	0	0	0	0	0	0
DM	β	-1	-1	0	0	0	0	0	0	0
SF	0	1	0	1	0	0	0	0	0	0
ASR	0	1	1	0	0	0	0	0	0	0
HPR	0	β	1	0	0	0	0	0	0	0
RAAR	0	1	β	0	0	0	0	0	$(1-\beta)$	0

Table 1: Specification of available iterative projective algorithms in the unified notation of Eq. 10 and the coefficients, $M_m,\ m = (1, 10)$, defined by Eq. refmultMC. ER: error reduction algorithm; BIO: basic input-output algorithm; HIO: hybrid input-output algorithm; DM: difference map algorithm; SF: solvent-flipping; ASR: averaged successive reflections; HPR: hybrid Projection Reflection; RAAR: relaxed averaged alternating reflectors. It is assumed here that $\mathbf{M}(\mathbf{x}_k) = -\mathbf{I}\mathbf{x}_k$ so that the changing role of the parameter β in the algorithms may be better displayed.

$$\begin{aligned}
\mathbf{v}_5(\mathbf{x}_k) &= \left[\hat{P}_F\hat{P}_S - \hat{P}_F\right]\mathbf{x}_k,\\
\mathbf{v}_6(\mathbf{x}_k) &= \left[\hat{P}_F\hat{P}_S - \hat{P}_S\right]\mathbf{x}_k,\\
\mathbf{v}_7(\mathbf{x}_k) &= \left[\hat{P}_F\hat{P}_S - \hat{I}\right]\mathbf{x}_k,\\
\mathbf{v}_8(\mathbf{x}_k) &= \left[\hat{P}_F - \hat{P}_S\right]\mathbf{x}_k,\\
\mathbf{v}_9(\mathbf{x}_k) &= \left[\hat{P}_F - \hat{I}\right]\mathbf{x}_k,\\
\mathbf{v}_{10}(\mathbf{x}_k) &= \left[\hat{P}_S - \hat{I}\right]\mathbf{x}_k.
\end{aligned}$$

This basis is linearly dependent since, for example, $\mathbf{v}_8 = \mathbf{v}_9 - \mathbf{v}_{10}$, so only linearly independent subsets of these vectors are selected in the construction of iterative projective algorithms. The vector $C(\mathbf{x}_k)$ is now defined to comprise the union of the vectors $\{\mathbf{v}_m(\mathbf{x}_k)\}$, and the matrix product $\mathbf{M}(\mathbf{x}_k)C(\mathbf{x}_k)$ is interpreted as meaning

$$\mathbf{M}(\mathbf{x}_k)C(\mathbf{x}_k) = \sum_{m=1}^{10} M_m(\mathbf{x}_k)\mathbf{v}_m(\mathbf{x}_k), \tag{18}$$

where $M_m(\mathbf{x}_k)$ are the elements of a diagonal matrix, whose effect is to multiply every element of the vector $\mathbf{v}_m(\mathbf{x}_k)$ by the scalar coefficient $M_m(\mathbf{x}_k)$, accumulating the sum to generate the output vector $\mathbf{M}(\mathbf{x}_k)C(\mathbf{x}_k)$. This formulation is sufficiently general that we may write any valid iterative projection scheme in the form defined by Eq. 10; a representative selection of these appears in 1. It is interesting that the vectors $M_5 - M_8$ and M_{10} do not appear in algorithms with satisfactory convergence characteristics.
Clearly, many related algorithms can be constructed following this pattern, but none can claim to be clearly superior to all others in every situation. In practice, we find that the hybrid input-output algorithm provides the most robust general-purpose workhorse for imaging, although we frequently alternate its use with error-reduction to enforce the support constraint strictly every few iterations. If only two-dimensional Fourier transforms are required, these procedures are so fast on modern computers that optimising computational efficiency is now seldom a critical consideration. It should also be noted that if these algorithms are executed using data containing noise, no two runs are likely to produce identical solutions, even if they have apparently converged. It is also quite likely that one will find that some runs of a particular algorithm will stagnate far from the solution, while others converge to a solution corresponding to a satisfactory fit to the data, if

a random start is adopted. One should adopt a statistical approach to analysing the set of "solutions" produced by these algorithms, rejecting stagnated solutions, and averaging over apparently converged solutions that contain small variations. Even if perfect noiseless data are employed, such as one might generate by simulation, *none* of the algorithms can be guaranteed to converge in all cases, within a "reasonable" number of iterations, if at all.
As Elser explains [8], we may not yet really claim that "an algorithm" exists for phase retrieval because we can place no useful bounds on the number of iterations that will be required to obtain a solution from an arbitrary starting guess; a truly satisfactory algorithm would be deterministic. It frequently happens, nevertheless, that a solution will emerge within, say, a few thousand iterations and, given the preceding analysis, that solution may be regarded as unique for all practical purposes. That a solution to a non-linear optimisation problem involving, typically, more than one million unknown function values may *ever* be obtained using such simple and computationally efficient algorithms is actually quite remarkable, and forms the practical basis of most modern approaches to diffractive imaging.

3 Structure of the Software

At the core of the NADIAsoftware is a set of C++ classes which compile to form a C++ library, `libNADIA.so` and `libNADIA.a`. It is anticipated that most users will access the functionality provided in our software by importing the library into their own C or C++ code. Several examples of how to include, link, and use the library within C/C++ code as discussed in the NADIA documentation in [18], and examples on how to run some examples are included in the following exercises.

We are aware that many users may wish to use our code within an interactive software environment that easily handles image manipulation and graphing. For this reason an interface has been written for the NADIAlibrary to IDL[23]. A large subset of the functionality of the C++ library is available in IDL. The same underlying library is executed for IDL and C/C++, therefore the performance is comparable. Examples IDL scripts have also been provided in the package.

For users with little or no software development experience, a command-line tool is provided to perform basic reconstruction using a read-in configuration file. Several command-line programs have been written to convert between tiff, hdf, ppm and binary image file formats, as well.

3.1 Software Design

The organisation of the C++ classes is shown in figure 1. The functionality that is common to more than one of the reconstruction classes is implemented in the abstract base class, `BaseCDI`. This class encapsulates operations such as applying the support constraint, scaling the intensity of the exit-surface wave in the detector plane to data, and setting the reconstruction algorithm.

The support constraint will be applied on the exit-surface-wave of a sample in the sample plane during reconstruction. If a user wishes to apply some other constraints on either the exit-surface-wave (plane-wave reconstruction) or the transmission function (`FresnelCDI`) they may do so using an instance of the `TransmissionConstraint` class. For example, in `FresnelCDI`, a `ComplexConstraint` may be defined for regions of the sample which are know to be homogeneous in material type.
The classes `PlanarCDI`, `FresnelCDI`, `FresnelCDI_WF`, `PartialCDI`, `PartCharCDI` and `PolyCDI` each inherit

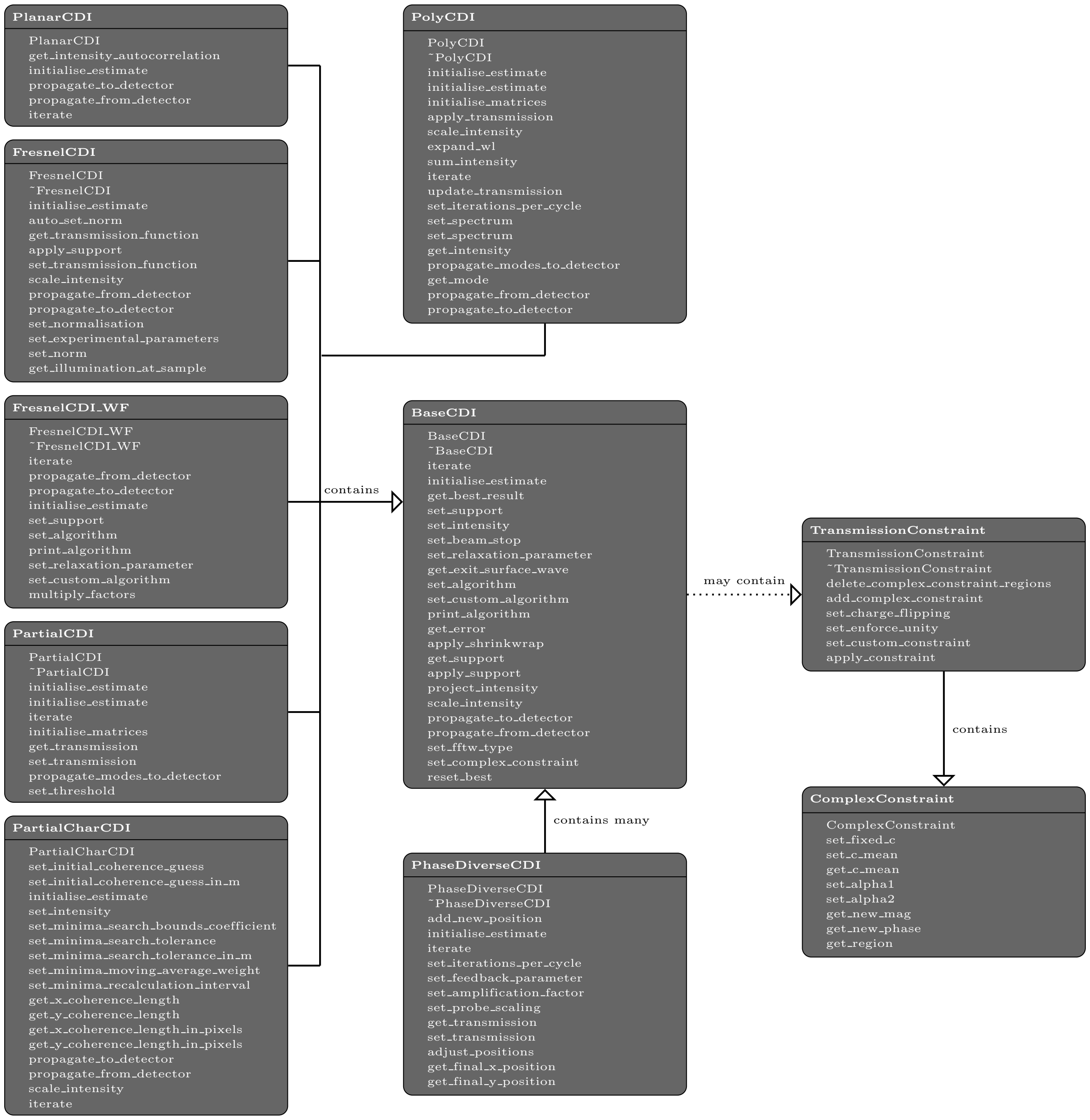


Figure 1: A class diagram showing the relationship between key classes in the software. The public methods are shown with the classes. The shaded region indicates the classes which are responsible for implementing various types of reconstructions. The non-shaded region shows classes which support these, and are generic to the reconstruction type. Note that some periphery classes are excluded from this diagram.

from `BaseCDI`. However, the code specific to each type of reconstruction (planar, Fresnel, Fresnel white-field, partially spatially coherent (modal), partially spatially coherent (Gaussian characterisation), and partially temporally coherent) is implemented within each class. The use of at least one instance of one of these class types is required for any reconstruction or simulation.
The algorithm for ptychographic [20] or phase-diverse [19] reconstruction is implemented in `PhaseDiverseCDI`. It takes as input, an number of objects derived from `BaseCDI` (i.e. objects of type `PlanarCDI` or `FresnelCDI`).
In constructing a generic algorithm in `BaseCDI` we make use of the factor that any algorithm can be decomposed into a linear set of the five operators, I, S, P, SP, PS as discussed in section 2.1.7. For each operator we create a 2D matrix of complex numbers at the start of each reconstruction which stores the exit-surface-wave estimate after the application of their corresponding operator. During each iteration, the updated estimate is copied to each matrix, the operator is applied and the results are recombined to form the updated exit-surface-wave.
To improve computational speed, we group the P and PS operators so that at most one Fast-Fourier transform is performed. This means that the if the result of P has been found, it is copied to the matrix holding the result for PS to which S is then applied. We could have implemented a similar grouping for S and SP, but as the computation expense of copying out-weighs that of applying the support constraint, we leave them as separate computations. In instances in which the coefficient to a term is zero, we neither compute that term nor allocate memory to the matrix which holds the result for that operator.
Finally, although the error reduction algorithm can be implemented in a such as those way described above, we instead treat error-reduction as a special case in order to improve performance. We simply apply PS directly on the matrix which holds the most recent ESW estimate. This results in faster run-time and lower memory usage.
More details on the technical aspect of the code design can be found within our Doxygen documentation on our website [18].

Finally, we also provide command-line tools which are driven by a text-based configuration file. These tools offer less flexible reconstruction options than the C++ library or IDL module, but are suitable for users without C/C++ or IDL experience who would like to get started quickly.

3.2 Installation

Our code uses the Fast Fourier Transforming library, fftw[12], the lapack[2] libraries for linear transforms, and the image and data I/O libraries libtiff[3] and hdf[6]. These libraries must be installed prior to installing our software.

In addition, users compiling from source code will require the g++ compiler and IDL users must have access to IDL. The most recent versions software that we have tested our code against are: fftw 3.3.3, libtiff 3.9.2, hdf 2.6-1, g++ 4.4.4.3-1 , IDL 8.1. Instructions describing how to install the prerequisite libraries for Ubuntu 12.10, Mac OSX 10.6 and Cygwin 1.7.17are given in in the documentation available on the NADIA website [18]. For Windows we distribute the compiled library files within our package.

The software is distributed as source code as well as a pre-compiled Windows 32-bit binary file. Windows users who wish only to use the IDL or command-line capabilities are encouraged to download the Windows binary file. Otherwise we recommend downloading and installing from the source.

Installing the package in a Linux environment is included as an exercise in this workbook, but for installation in a Windows or Mac environment, please see the documentation on the NADIA website [18].

3.3 NADIA Examples

The package contains of a set of examples that demonstrate how to use the library and tools for various reconstructions and simulations. They are located in the `examples` and `/interfaces/idl` subdirectories of the package, and will be compiled automatically after installation. Table 2 lists the example filenames. In many cases we show how the same reconstruction can be achieved using any of the C++ library, the IDL module and the command line tools. Simulation examples are given for C++ only, although it is also possible to perform simulations in IDL. These examples will not be directly used in the following exercises, but a description of each of the examples is provided here for reference.

C++/C	IDL	Command-line tool
PlanarCDI_example.c	planar_example.pro	planar_example.config
PlanarCDI_simulation_example.c		
FresnelCDI_WF_example.c	fresnel_example.pro	
FresnelCDI_example.c	fresnel_example.pro	fresnel_example.config
FresnelCDI_simulation_example.c		
ComplexConstraint_example.c		
PhaseDiverse_example.c	phase_diverse_example.pro	
PartialCDI_example.c	partial_example.pro	
PartialCharCDI_example.c	partialchar_example.pro	
PolyCDI_example.c	poly_example.pro	

Table 2: Example code filenames provided within the NADIApackage and their corresponding. C/C++ files will be automatically compiled during installation and can be run by typing `./<file name>.exe` in the `examples` subdirectory, where `<filename>` corresponds to the file name in the table without the .c extension.The IDL code can be run from the command line by changing to the `NADIA/interfaces/idl` directory, and executing `idl/<filename>.pro`. The command line tool examples can be run by making sure that the `NADIA/bin` directory is included in the path, and then by executing `CDI_reconstruction.exe <filename>.config` for the planar and Fresnel examples.

3.3.1 Example: Plane-wave

The plane-wave diffraction data published in [24] is reconstructed using a combination of the hybrid input-output (HIO) and the error-reduction algorithm. In the default example 50 ER iterations are followed by 100 HIO iterations and another 50 iterations of ER are performed for 2 cycles. Every 50 iterations the shrinkwrap algorithm is applied and every 10 iterations the current estimate is output as a .ppm.
Using a previously reconstructed image of the same sample we also demonstrate how to simulate a diffraction pattern for plane-wave CDI. The image is propagated to the detector to find the diffraction pattern and a threshold is applied in order to make the diffraction pattern more closely resemble the reading the readings found in a detector. The diffraction pattern is saved, and a reconstruction performed.

3.3.2 Example: Fresnel

We provide four examples for Fresnel experiments:

- One frame of the phase-diverse data from [19] is used is this example. We reconstruct the phase of the white field and save the result to file.
- We then demonstrate how to reconstruct the sample exit-surface wave and transmission function using this white-field. The Fresnel error-reduction algorithm is used. The phase and magnitude of the transmission function are output to file.

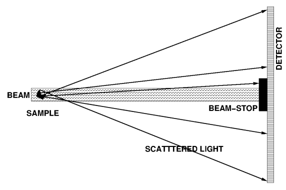


Figure 2: Schematic representation of the configuration of a plane-wave coherent-diffractive imaging experiment.

- We build on this example by demonstrating how complex and other constraints on the transmission function can be included in a reconstruction.
- Finally, we provide an example of Fresnel simulation. This makes use of the object from the plane-wave example and the white-field reconstruction in the Fresnel example.

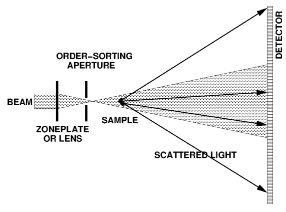


Figure 3: Schematic representation of the configuration of a Fresnel coherent-diffractive imaging experiment.

3.3.3 Example: Phase-diverse/Ptychographic

One example for phase-diverse/ptychographic reconstruction is given; we show how the data published in [19] can be reconstructed. It consists of seven Fresnel diffraction images of the same object. For each image, the sample position has been altered with respect to the focal point position and/or transverse to the beam direction.

3.3.4 Example: Partially Spatial Coherence

Example code for the reconstruction of planar partially spatially coherent data using two different methods is provided. In the first case, the reconstruction is performed as described in [24]. The data and modal expansion from that paper is used to replicate the reconstruction. Normalised Legendre polynomials are used to expand the orthonormal, mutually incoherent modes. The example defaults to 15 cycles of 50 ER, 50 HIO and then 50 ER with the shrinkwrap algorithm applied every 150 iterations. The simulation example takes the same sample image as the planar illumination example, expands the modes in the Legendre space, and propagates the modal intensities to the detector to find the diffraction pattern of each mode. The modes are added together and a threshold applied to the diffraction pattern to create a realistic detector image.

An alternative, non-modal, approach to partial spatial coherence is described in [7] for the case of a Gaussian mutual optical intensity (MOI) function. This method has the advantage that relatively little needs to be known about the beam's characteristics if it can be assumed to have a Gaussian coherence function. This algorithm finds lateral coherence lengths l_x and l_y that minimise error in the reconstruction, thus enabling the beam's coherence to be characterised as reconstruction is performed. *coherence length* is defined, in the context of this algorithm, as the standard deviation of the Gaussian MOI function.
In NADIA, this algorithm is implemented in the `PartialCharCDI` class. The operation of this class is very similar to `PlanarCDI` with a few additional parameters. At initialisation, the sample-detector distance, beam energy, and the size of the detector pixels must be specified[1] - these values allow conversion between coherence length in detector-plane-pixels and object-plane-meters.
Optionally, additional parameters may be set, with the methods described below, to control the initial-guess, and search parameters for the optimising coherence lengths.

3.3.5 Example: Partial Temporal Coherence

Example code for the simulation and reconstruction of planar partially spatially coherent data is provided. Simulated data is provided for the reconstruction. The method used in this reconstruction is described in [1]. The simulated beam was generated using SPECTRA [21] using parameters representative of the beamline at the Australian Synchrotron. The beam profile was output as a text file that can be read in by NADIA. The same image as the planar simulation was propagated to the detector and the image was rescaled according to the sampled wavelengths. The user can also define their own beam profile in the form of a 2 dimensional array.

[1]These parameters are not strictly necessary if you don't want to set any other options in meters (ie. if you are using the default initial estimates and search parameters) and if you don't want to get optimal coherence values in meters (just in pixels).

4 Exercises

Exercise #1: Preparing Your System for NADIA

As mentioned in section 3.2, the NADIA Package uses fftw, libtiff, lapack, and HDF4 libraries. You need to make sure that these are installed before you try to configure and compile NADIA. On an Ubuntu system, you would type:

```
dpkg -s liblapack-dev
```

to check whether the development library for LAPACK is installed.
Do this for liblapack-dev, libtiff-dev and libfftw3-dev. HDF4 is not managed through the package manager, and so will need to be installed separately following the instructions at the HDF website[6].
If these are all installed, you are ready to proceed. Download the package from the COECXS website, and unzip it by typing

```
tar -xvf NADIA.0.5.tar.gz
```

then change directory in to the NADIA folder.

Exercise #2: Configuring Your Install

Before you install NADIA, you need to let the package know which options you want. Type

```
./configure --help
```

To see what options are available to you. One interesting option is `--enable-double-precision` *use double rather than single precision in the reconstructions*, which will tell the compiler that you would like increase the space in memory used to store values and, as a result, increase the precision with which you can reconstruct images. In the case of this work book, however, the default options should be adequate. With that in mind, type:

```
./configure
```

If you would like to change you compilation options at any time, you need to clear the results of the configuration so that you can start over. If you need to do this type `make bare`. Otherwise, proceed to the next exercise.

Exercise #3: Compiling your Install

The directory structure of the NADIA package is shown below.

doc - Doxygen documentation and manual

examples - Reconstruction code to demonstrating how to use the software

include - The NADIAheader files

interfaces - Wrapper code for programming languages other than C/C++

idl - IDL interface for the library and some examples

lib - The directory for the compiled code

bin - The command line tool binaries

src - The NADIAlibrary source code

tools - Source code for the command-line tools

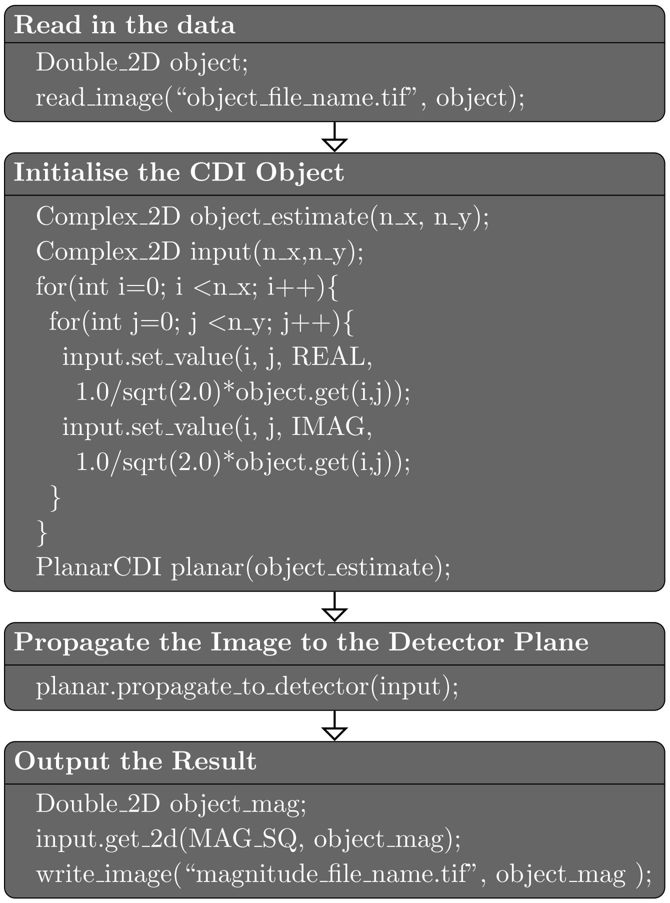
This step takes the .c and .c++ files and turns *compiles* them in to executable .exe files.
To do this type:

```
make
```

Each time you change a file, will need to compile the files again to create new .exe files.
After compilation the `bin` and `lib` directories will be filled with the executable command-line tools, and the compiled library. You should now have some executable examples in the example folder. Feel free to run these, or continue on to the next exercise.

Exercise #4: Simulating a Diffraction Pattern

The NADIA simulates in the following way:



Read in the data

```
Double_2D object;
read_image("object_file_name.tif", object);
```

A 2D array is created to hold the object data. The array is then filled.

Initialise the CDI Object

```
Complex_2D object_estimate(n_x, n_y);
Complex_2D input(n_x,n_y);
for(int i=0; i <n_x; i++){
 for(int j=0; j <n_y; j++){
   input.set_value(i, j, REAL,
     1.0/sqrt(2.0)*object.get(i,j));
   input.set_value(i, j, IMAG,
     1.0/sqrt(2.0)*object.get(i,j));
 }
}
PlanarCDI planar(object_estimate);
```

A Complex 2D object is created to hold the evolving result from the simulation and is filled with the object data. A CDI object (in this case a Planar CDI object) is created.

Propagate the Image to the Detector Plane

```
planar.propagate_to_detector(input);
```

The object data is propagated to the detector plane.

Output the Result

```
Double_2D object_mag;
input.get_2d(MAG_SQ, object_mag);
write_image("magnitude_file_name.tif", object_mag );
```

The magnitude of the propagated object is output in to a Double_2D, and then written to file. In this case it is written as a tif.

Download the Workbook folder and untar it in to the NADIA folder, and change directory in to it. Read through `PlanarCDI_simulation_example.c` and use it to simulate some output data. To do this, edit it to use `image_files/object.tiff` as the data, and the file name of your choice as the output. Copy `Makefile` from the `examples` folder, and edit it to compile only `PlanarCDI_simulation_example.c`. Type `make` to compile your edited file.

All of the different reconstruction types need to be initialised with the diffraction data, support data and initial estimate (whether a random initialisation or the result of a previous reconstruction). Additionally, Fresnel and partially spatially coherent reconstructions require parameters related to the detector setup, and partially temporally coherent reconstruction requires the energy profile of the beam used. The requirements of each class can be found in the Doxygen documentation.

Within our library, data is stored and passed in transient objects of type `Double_2D`, a two dimensional array of floating point numbers (for scalar fields), and `Complex_2D` a two dimensional array of single precision or double precision (decided by the user) numbers in the form of a `fftw_complex` for the representation of complex fields.

Functions are provided to convert hdf [6], tiff or ppm images into NADIA's native format and to output images as a grey scale tiff or a 16-bit ppm image [2]. In addition data can be read and stored as a binary file of doubles.

[2]Note that pixel values will be scaled in this case and the values in the `Double_2D` object will be different if the image file is re-read into the software

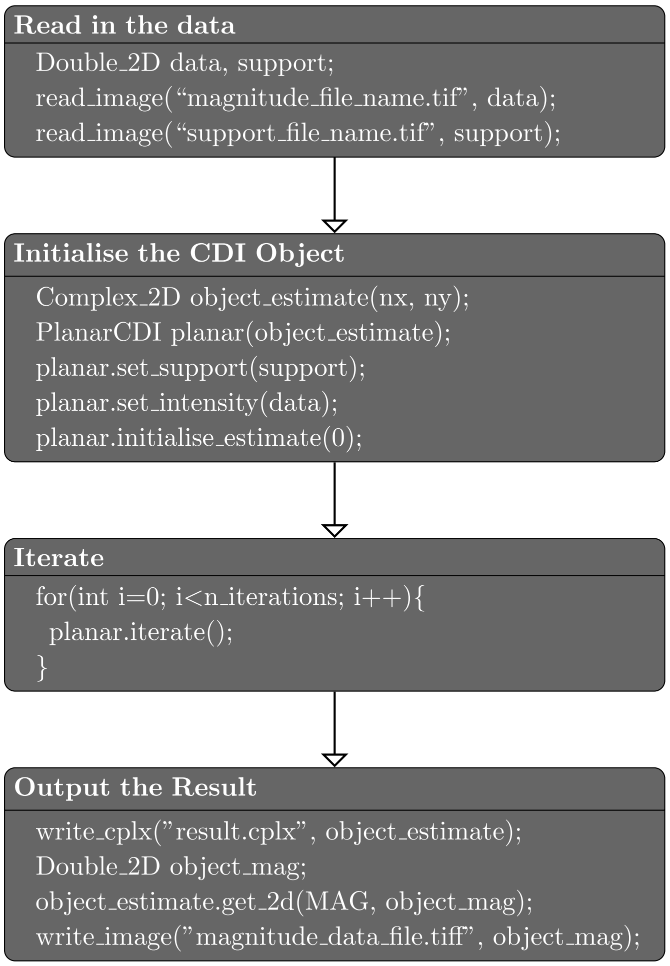
Exercise #4: Simulating a Diffraction Pattern continued on next page. . .

This may be useful in situations where higher precision is required. Similarly, two dimensional arrays of complex numbers, such as those produced in the reconstruction, may be stored as a binary file of fftw [12] complex number types.

Take this time to have a look at the object and output files.

Exercise #5: Reconstructing a Simulated Diffraction Pattern

The NADIA package reconstructs in the following way:



Read in the data

```
Double_2D data, support;
read_image("magnitude_file_name.tif", data);
read_image("support_file_name.tif", support);
```

2D arrays are created to hold the detector data and support information. The arrays are then filled, in this case, from .tif's.

Initialise the CDI Object

```
Complex_2D object_estimate(nx, ny);
PlanarCDI planar(object_estimate);
planar.set_support(support);
planar.set_intensity(data);
planar.initialise_estimate(0);
```

A Complex 2D object is created to hold the evolving result from the simulation and is filled with the object data. A CDI object (in this case a Planar CDI object) is created. The detector data, support information and initial estimate are initialised.

Iterate

```
for(int i=0; i<n_iterations; i++){
  planar.iterate();
}
```

The object estimate is propagated to and from the detector plane according to the selected algorithm.

Output the Result

```
write_cplx("result.cplx", object_estimate);
Double_2D object_mag;
object_estimate.get_2d(MAG, object_mag);
write_image("magnitude_data_file.tiff", object_mag);
```

The complex object estimate is written to file, as is the magnitude of the object estimate.

Read through `PlanarCDI_example.c` and use it to begin reconstruction of the data you simulated in the above example. To do this, use the output file from the previous example as the input data. Use `image_files/planar_support.tiff` as the support. Copy the `Makefile` from the examples folder. Edit the `make` file to compile only `PlanarCDI_example.c`, and type `make`.

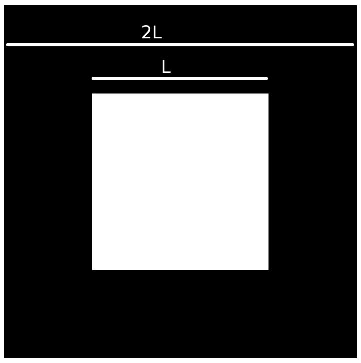


Figure 4: The length of the support area must not exceed L in any direction. The object is required to be contained entirely within the white area.

The support is a black and white image that is used by the software to constrain the possible image. The image must be contained entirely within the white area. Due to sampling constraints, the length of this area must be no more than half of the length of the total sampled area in any direction, as per Figure 4.

Exercise #6: Outputting Images

You may want to view the output of the reconstruction as it is running. Copy and past the following into the bottom of `for` loop:

```
if(i%10==0){

Double_2D result(nx, ny); \\create an Double_2D to hold the result
object_estimate.get_2d(MAG,result); \\put the magnitude of object_estimate in to Double_2D

    ostringstream temp_str ( ostringstream::out ); \\Create a string to hold the file name
    temp_str << "planar_example_iteration_"  << i << ".tiff";  \\ Fill the string with the
     \\name of the file
    write_image(temp_str.str(), result); \\Write the result to the named file

}
```

If you don't understand a lot of C++, this may seem a bit tricky to follow, but don't worry about the string stuff too much. The important commands are the copying of the magnitude in to the result Double_2D, and the write_image command. This block of code will output the magnitude of the current object estimate every 10 iterations. By changing `MAG` to `PHASE` you can output the phase of the estimate. Try it now. Also try to change the number of iterations between outputs.

Exercise #7: The Shrinkwrap Algorithm

You may notice that your reconstruction is reconstructing a bit too slowly. In order to speed it up we will apply the shrinkwrap algorithm. The shrinkwrap algorithm convolves the reconstructed image with a user-defined Gaussian, and then finds a new support by applying a threshold to the resulting convolution [22]. The shrinkwrap algorithm is applied by calling

```
planar.apply_shrinkwrap(sigma, threshold);
```

where `sigma` is a float > 1.0, and the standard deviation of the Gaussian in pixels, and `threshold` is a float $0.0 < threshold < 1.0$ denoting the minimum value a pixel must have as a proportion of the value of the maximum pixel to be included in the new support. In the examples in the example folder I use values of $sigma = 2.0$ and $threshold = 0.1$. Modify your code to apply the shrinkwrap algorithm every 50 iterations.

Exercise #8: Saving and Loading from a Previous Reconstruction

The line

```
write_cplx("result.cplx", object_estimate);
```

writes the current object estimate phase and intensity to the file `result.cplx`.
The line

```
planar.initialise_estimate(0);
```

initialises the object estimate with a random intensity and phase. The random number generator in this case is initialised with the seed 0. If you want to read in the result of a previous reconstruction, comment this line, and replace is with

```
//planar.initialise_estimate(0);
read_cplx("result.cplx", object_estimate);
```

Don't forget to type make again to compile any changes.

Exercise #9: Changing Algorithms

You may notice now that your reconstruction has gotten further along and is approaching an image, but has now reached a point where it is no longer improving. This is called *stagnation.* In order to combat this we will try switching algorithms. You can change the reconstruction algorithm by using

```
planar.set_algorithm(algorithm);
```

Where *algorithm* can be replaced with any of

- ER - error reduction
- BIO - basic input-output
- BOO - basic output-output
- HIO - hybrid input-output
- DM - difference map
- SF - solvent-flipping,
- ASR - averaged successive reflections
- HPR - hybrid projection reflection
- RAAR - relaxed averaged alternating reflectors

Or you can set a custom algorithm using

```
planar.set_custom_algorithm(0.5, 0, -1 ,0, -1,0,0,0,0,0);
```

following the numbering convention in table 1.

Create a second `for` loop in your code following the error reduction for loop, and set it to use hybrid input-output algorithm for 50 iterations. Don't forget to include the shrinkwrap and output code. Think about how you will manage the iterators.

Exercise #10: Partial Spatial Coherence

Look at `PartialCDI_simulation_example.c` and run the matching executable. This example simulates a Partially Spatially coherent beam incident on the same example as for the previous case. It assumes a beam with the properties as in [25], with

- Coherence lengths: $lcx = 13.3 \times 10^{-6}$m, and $lcy = 40.0^{-3}$m.
- Pixel size of detector $= 13.5 \times 10^{-6}$m in the x and y direction.
- Number of Modes in one dimension = 7 making a total of 49 modes.
- Distance between the detector and sample = 1.4m.
- Number of Legendre Polynomial = 8. *The number of polynomials must be greater than the number of modes in either direction.*
- Energy of the beam = 1400.0eV.

Read through this example and use it to simulate some partially spatially coherent data. Don't forget to include `PartialCDI_simulation_example.c` in the `make` file.

Exercise #11: Reconstructing Partially Coherent Data

Each of the different NADIA CDI classes are built based on the same basic `BaseCDI` class. As a result, they can be used in the same basic way. In this exercise, you will copy your `PlanarCDI_example.c` from the previous exercises, and modify it to reconstruct some real, partially coherent data. You will need to change

- The included libraries at the top of the file `#include <PartialCDI.h>`
- The PlanarCDI objects to PartialCDI objects.
- The way the PartialCDI object is initialised
- The support and data that you are using (`image_files/part_data.dbin` and `image_files/part_support.tiff`)
- The number of pixels in the object (nx and ny should be 2048)
- The `make` file
- And don't forget that this is a new reconstruction, so you will want a new, random initial guess. Don't try to read in your old one.

Exercise #12: Reconstructing Polychromatic or Partially Temporally Coherent Data

There is a `PolyCDI_simulation_example.c` in this workbook folder that you are free to examine and run, but the simulated data based on that example is available at `image_files/poly_sim_intensity.tiff` and the matching SPECTRA file is at `image_files/spectrum.txt`. This file is the text output of SPECTRA [21] based on the properties of the Australian Synchrotron.

A `PolyCDI` is initialised in the same way as a `PartialCDI`, with the additional step of initialising the spectrum of the light source. A special function has been written to read in a spectrum in the form of a text file such as that output by SPECTRA, but a spectrum can also be passed in as a `Double_2D` with the wavelength in the first column and the relative weight of that wavelength in the second column. This means that you could write your own function to read in spectra, or even create your own using a simple mathematical function and a for loop.

Run and reconstruct the Polychromatic example. Feel free to try tweaking the methods used and number of iterations to speed up each method. Don't forget to save your output. If you wanted, you could simulate and reconstruct your own polychromatic data based on your own, self-generated spectrum.

References

[1] Brian Abbey, Lachlan W. Whitehead, Harry M. Quiney, David J. Vine, Guido A. Cadenazzi, Clare A. Henderson, Keith A. Nugent, Eugeniu Balaur, Corey T. Putkunz, Andrew G. Peele, G. J. Williams, and I. McNulty. Lensless Imaging Using Broadband X-Ray Sources. *Nature Photonics*, 5(7):420 – 424, 2011.

[2] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: A Portable Linear Algebra Library for High-Performance Computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, Supercomputing '90, pages 2–11, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[3] Adobe Developers Association. *TIFF Revision 6.0.* Adobe Systems Incorporated, 1585 Charleston Road, P.O. Box 7900, Mountain View CA 94039-7900, June 1992.

[4] Heinz H Bauschke, Patrick L Combettes, and D Russell Luke. Phase Retrieval, Error Reduction Algorithm, and Fienup Variants: a View From Convex Optimization. *JOSA A*, 19(7):1334–1345, 2002.

[5] Heinz H Bauschke, Patrick L Combettes, and D Russell Luke. Hybrid Projection–Reflection Method for Phase Retrieval. *JOSA A*, 20(6):1025–1034, 2003.

[6] Copyright 1988-2006 by the Board of Trustees of the University of Illinois. NCSA Hierarchical Data Format (HDF) Software Library and Utilities.

[7] Jesse N. Clark and Andrew G. Peele. Simultaneous Sample and Spatial Coherence Characterisation Using Diffractive Imaging. *Applied Physics Letters*, 99:3, 2011.

[8] Veit Elser. Phase Retrieval by Iterated Projections. *JOSA A*, 20(1):40–55, 2003.

[9] Veit Elser, I Rankenburg, and P Thibault. Searching with Iterated Maps. *Proceedings of the National Academy of Sciences*, 104(2):418–423, 2007.

[10] James R Fienup. Reconstruction of an Object From the Modulus of its Fourier Transform. *Optics Letters*, 3(1):27–29, 1978.

[11] James R Fienup et al. Phase Retrieval Algorithms: A Comparison. *Applied Optics*, 21(15):2758–2769, 1982.

[12] Matteo Frigo and Steven G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[13] RW Gerchberg. A practical Algorithm for the Determination of Phase From Image and Diffraction Plane Pictures. *Optik*, 35:237, 1972.

[14] HB Keller. Numerical methods for two-point boundary-value problems, blaisdell publ. *Co., Waltham, Massachusetts*, 1968.

[15] Herbert Bishop Keller and Eugene Isaacson. Analysis of numerical methods. *J. Wiley & Sons, New York*, 1966.

[16] Stefano Marchesini. Invited Article: A Unified Evaluation of Iterative Projection Algorithms for Phase Retrieval. *Review of Scientific Instruments*, 78(1):011301–011301, 2007.

[17] Stefano Marchesini. Phase retrieval and saddle-point optimization. *JOSA A*, 24(10):3289–3296, 2007.

[18] ARC Centre of Excellence for Coherent X-Ray Science. Nadia Web Page. http://www.coecxs.org/joomla/index.php/research-and-projects/nadia-software-project.html

[19] Corey T. Putkunz, Jesse N. Clark, David J. Vine, Garth J. Williams, Mark A. Pfeifer, Eugeniu Balaur, Ian McNulty, Keith A. Nugent, and Andrew G. Peele. Phase-diverse coherent diffractive imaging: High sensitivity with low dose. *Phys. Rev. Lett.*, 106:013903, Jan 2011.

[20] Corey T. Putkunz, Adrian J. D'Alfonso, Andrew J. Morgan, Matthew Weyland, Christian Dwyer, Laure Bourgeois, Joanne Etheridge, Ann Roberts, Robert E. Scholten, Keith A. Nugent, and Leslie J. Allen. Atom-scale ptychographic electron diffractive imaging of boron nitride cones. *Phys. Rev. Lett.*, 108:073901, Feb 2012.

[21] T. Tanaka and H. Kitamura. Spectra: A Synchrotron Radiation Calculation Code. *Journal of Synchrotron Radiation*, 8:1121-1228, 2001

[22] Stefano , H He, Henry N Chapman, Stefan P Hau-Riege, A Noy, Malcolm R Howells, U Weierstall, and John CH Spence. X-Ray Image Reconstruction from a Diffraction Pattern Alone. *Physical Review B*, 68(14):140101, 2003.

[23] Exelis Visual Information Solutions IDL (Interactive Data Language).

[24] L. W. Whitehead, G. J. Williams, H. M. Quiney, D. J. Vine, R. A. Dilanian, S. Flewett, K. A. Nugent, A. G. Peele, E. Balaur, and I. McNulty. Diffractive Imaging Using Partially Coherent X Rays. *Phys. Rev. Lett.*, 103:243902, Dec 2009.

[25] LW Whitehead, GJ Williams, HM Quiney, DJ Vine, RA Dilanian, S Flewett, KA Nugent, AG Peele, E Balaur, and I McNulty. Diffractive Imaging Using Partially Coherent X Rays. *Physical Review Letters*, 103(24):243902, 2009.